Noah ALVES 21105811

Alejandra MORALES SAUCEDO 21214631

Rapport Réorganisation d'un réseau de fibres optiques

Année 2023-2024 LU2IN006 - Structures de Données

Description du projet

Le projet a porté sur la réorganisation d'un réseau de fibres optiques d'une agglomération grâce à différentes structures de données telles qu'une liste chaînée, une table de hachage, un arbre quaternaire et un graphe.

Le projet se divise en deux parties :

la reconstitution du réseau et la réorganisation du réseau.

Le réseau de fibres se compose d'un ensemble de câbles, chaque câble comportant un ensemble de fibres optiques qui relient des clients.

Pour la première partie, la reconstitution du réseau consiste à regrouper l'ensemble des fibres optiques de tous les opérateurs qui constituent le réseau de l'agglomération afin d'avoir un plan complet du réseau.

Pour la deuxième partie, la réorganisation du réseau consiste à répartir les *chaînes* de tronçons de fibres qui relient une pair de clients afin d'éviter les problèmes de sur-exploitation et longueurs excessives des chaînes.

Un câble du réseau est un fourreau qui contient exactement *gamma* > 0 fibres optiques. Les câbles relient deux points du plan, soit des clients ou des concentrateurs qui eux permettent de relier des tronçons de fibres. Ces tronçons, reliés par un même concentrateur forment une châine dans le réseau. Une chaîne relie deux points clients qu'on appelle une *commodité*, les extrémités de cette chaîne.

Description des structures manipulées, description globale de votre code

Partie 1 : Lecture, stockage et affichage des données

Cette partie est composée de l'exercice 1, cet exercice a pour but de reconstruire une instance d'une structure Chaines à partir d'un fichier, ou bien d'écrire le contenu d'une Chaines sur un fichier en respectant un format donné. Nous avons trois fichiers utilisés pour cet exercice : **Chaine.c, Chaine.h et ChaineMain.c** Nous pouvons éventuellement avoir une représentation graphique des instances grâce à la fonction **afficheChaineSVG**, elle permet de créer un fichier SVG en HTML et qui sera lu depuis un explorateur internet.

Partie 2 : Reconstitution du réseau

Cette partie est composée des exercices 2,3,4,5 et 6.

Dans l'exercice 2, nous devons reconstruire le réseau à partir de la liste chaînée. Le fichier **Reseau.c** contient l'ensemble des fonctions qui permettent de reconstruire le réseau à partir de la liste chaînée dont la fonction principale *reconstitueReseauListe(Chaines* C);*. Le fichier **Reseau.h** contient l'ensemble des structures et définitions des fonctions utilisées.

On crée aussi un programme main **ReconstitueReseau.c** qui permet de choisir en paramètre la méthode que l'on désire utiliser pour reconstituer le réseau avec une structure spécifique sur un fichier .cha donné en paramètre.

Dans l'exercice 3, on souhaite construire des méthodes afin de manipuler et d'afficher la structure Reseau. On calcule le nombre de commodités et de liaisons d'un réseau, on écrit le contenu du réseau sur un fichier et on l'affiche grâce à la fonction *afficheReseauSVG(Reseau* R, char *nomInstance);* qui crée un fichier SVG en html pour visualiser le réseau.

Dans l'exercice 4, nous devons reconstruire le réseau à partir d'une table de hachage. Nous avons créé un fichier **Hachage.c** qui va contenir l'ensemble des programmes qui nous est utile tels que la création d'une clé et une fonction de hachage. On a aussi un fichier **Hachage.h** pour les signatures et la structure TableHachage.

Dans l'exercice 5, la reconstruction du réseau se fait avec un arbre quaternaire. Le fichier **ArbreQuat.h** contient la structure du noeud de cet arbre, ainsi que l'ensemble des signatures des fonctions qu'on peut retrouver dans le fichier **AbreQuat.c**.

Dans l'exercice 6, on crée un fichier **temps_de_calcul.c** afin de comparer les temps de calcul obtenus avec les 3 structures données. On écrit une fonction *Chaines* generationAleatoire(int nbChaines,int nbPointsChaine,int xmax,int ymax);* qui permet de créer des chaînes de points avec des valeurs passées en paramètre. On renvoie les résultats sur des fichiers nommés **temps_de_calcul_ha.txt** et **temps_de_calcul_lc.txt** afin d'utiliser gnuplot pour créer des graphes correspondant à ces valeurs.

On utilise cette fonction afin de déterminer la structure la plus optimale pour reconstituer le réseau en faisant varier le nombre de chaînes de 500 à 5000 et la taille de la table de hachage. Des fichiers **temps_de_calcul_ha.txt** et **temps_de_calcul_lc.txt** sont crées afin de réaliser des graphes pour comparer les temps de calcul.

Partie 3 : Optimisation du réseau

Dans l'exercice 7, notre but est d'optimiser l'utilisation des fibres optiques du réseau. On crée une structure graphe définie dans le fichier **Graphe.h** et dans le fichier **Graphe.c** on écrit une fonction *int reorganiseReseau(Reseau* r);* qui permet de calculer la plus courte chaîne pour chaque commodité grâce à une matrice.

Fichiers et fonctions hors sujet

Pour tous les fichiers .c qui utilisent une fonction de type ReconstitueReseau[nom_structure] nous avons utilisé une fonction *void ajouter_voisin(Noeud* n, Noeud * voisin); qui nous permet de simplifier la tache pour reconstituer le réseau. Pour chaque fichier, nous avons aussi créé des fonctions afin de libérer la mémoire allouée pour les différentes structures.

Le fichier **graphiques.txt** contient les instructions pour lancer gnuplot et créer les fichiers .ps qui affichent les temps de calcul des structures.

Le fichier **Makefile :** Pour compiler les fichiers correspondant à une structure ou exercice spécifique, il suffit de choisir une structure parmi les suivantes : *[chaine, res, hachage ,arbrequat, ReconstitueReseau, temps_de_calcul, graphe]*

Tapez les instructions suivantes sur votre terminal:

make <structure_choisie>

Puis pour exécuter le programme :

./<structure_choisie>

Si vous voulez compiler toutes les structures ensemble, il suffit de faire :

make all

Dans le cas o`u un exécutable aura besoin d'un fichier ou entier passé en paramètre, des instructions seront affichées sur le terminal.

Réponses aux questions

Exercice 4, question 2 Suite à notre test sur les points (x,y) avec *x* un entier allant de 1 à 10 et *y* un entier allant de 1 à 10, nous constatons que plusieurs cléfs ont la même valeurs, soit 88 cléfs sur 100 sont répétées au moins une fois sur notre test.
De ce fait, la fonction *cle (x,y)* nous semble pas appropriée, car elle peut engendrer plusieurs collisions. L'unicité de la cléf de hachage est essentielle pour diminuer les problèmes de collisions.

Exercice 6, question 1 Quand on exécute les trois fonctions de reconstruction avec le temps de calcul pour les instances fournies, on observe que lorsqu'on utilise un nombre de chaînes très petit, les temps de calcul pour les 3 structures sont plutôt similaires.

Pour une instance plus grande, si la table de hachage est de taille inférieure au nombre de chaînes, son temps de calcul ressemble à celui de la liste chaînée, soit 40 fois plus long que celui de l'arbre quaternaire. Cependant, si la table de hachage est de taille supérieure ou égale au nombre de chaînes, sont temps de calcul se rapproche à celui de l'arbre

Exercice 6, question 4 Voir la section : Analyse commentée des performances de nos programmes

Exercice 7, question 5 En testant la fonction reorganiseReseau sur les trois instances données, nous avons remarqué qu'il existe au moins une arête dont la valeur dépasse le gamma. Pour améliorer cette fonction, il serait judicieux de prendre en compte les situations où une arête se rapproche du gamma ou l'atteint.

Lorsqu'une arête s'approche du gamma, il pourrait être intéressant de rechercher un autre chemin pour les autres arêtes, même si ce chemin n'est pas le plus rapide. De plus, nous pourrions stocker les arêtes qui ne sont plus accessibles (c'est-à-dire celles qui ont été parcourues trop de fois) afin de trouver une alternative.

Cela pourrait être une solution pour améliorer notre programme et réduire le nombre d'arêtes qui dépassent le seuil gamma.

Description des jeux d'essais

Exercice 1 : Exemple d'usage : **./ChaineMain nom_fichier.cha** Une structure Chaine sera créée et l'utilisateur pourra voir sur le terminal la longueur totale des chaînes et le nombre total de point. On lui demande s'il veut créer ou non un autre fichier avec cette chaine.

Exercice 2 : La question 2.3 nous demande d'écrire un main **ReconstitueReseau.c** Exemple d'usage : **./ReconstitueReseau fichier.cha n** (n un entier entre 1 et 3 pour choisir la méthode). 1er méthode : Liste chainéé. 2ème méthode : Table de Hachage 3eme méthode : Arbre quaternaire.

Pour chacune des méthodes, un fichier .txt est créé dans le répertoire de l'utilisateur qui a pour but de voir l'ensemble des sommets, l'ensemble des liaisons et l'ensemble des commodités. Ainsi qu'un fichier .html pour visualiser le réseau.

Exercice 6 : La question 6.1 nous demande d'écrire un main qui exécute automatiquement les trois fonctions de reconstruction et qui calcule uniquement leur temps de calcul. Ce main est écrit dans le fichier **temps_de_calcul.c**. Exemple d'usage : **./temps_de_calcul.c nom_fichier.cha** A chaque fois que l'utilisateur va faire cette commande, cela va écrire les temps de calcul pour chacune des méthodes.

Exercice 7 : Pour tester la fonction reorganisationReseau, nous avons décidé de créer un fichier **mainG.c** qui va demander à l'utilisateur d'entrer un fichier .cha et un numéro de méthode. Le numéro de méthode permet de savoir avec quelles méthodes l'utilisateur souhaite réorganiser le réseau.

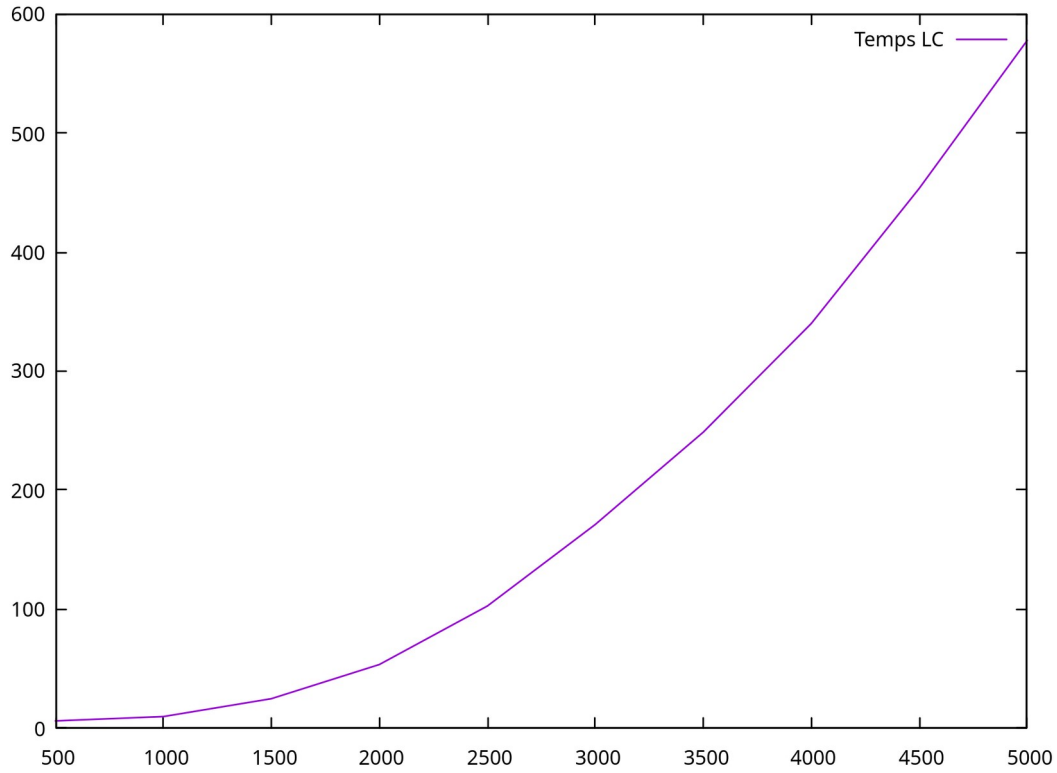
Ensuite, nous appliquons la fonction reorganisation sur le réseau qui vient d'être créé, afin de déterminer si nous parcourons chaque chaîne un nombre de fois supérieur au gamma ou non. Les arêtes qui dépassent le gamma seront affichées dans le terminal, ainsi que le nombre d'arêtes qui le dépassent.

Exemple d'usage: **./graphe nom_fichier.cha n** (n un entier entre 1 et 3 pour choisir la méthode)

Analyse commentée des performances de nos programmes

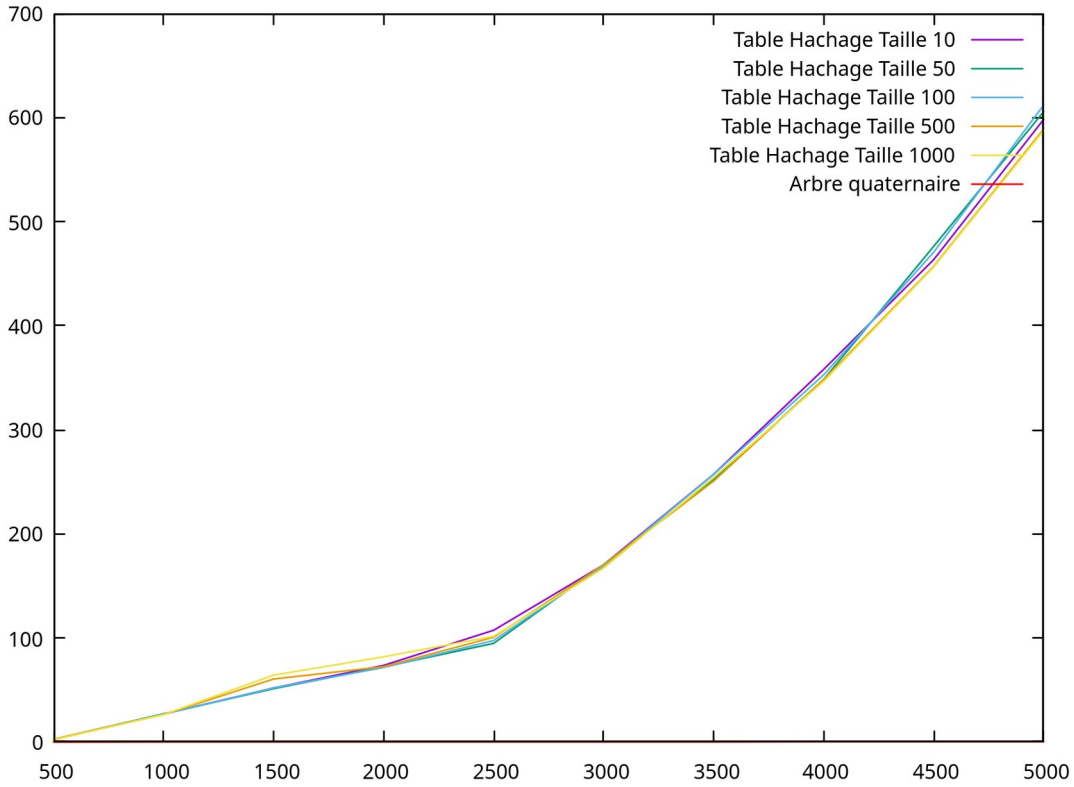
Quand on exécute le code de la question 6.3 qui renvoi les temps de calcul pour la fonction *ReconstitueReseau* de chaque structure : la liste chaînée, la table de hachage et l'arbre quaternaire. On fait varier la taille de la table de hachage avec les valeurs de M suivantes :
10, 50, 100, 500, 1000.

En observant les temps de calculs stockés dans les fichiers *temps_calcul_ha.txt* et *temps_calcul_lc.txt* on remarque que la reconstitution du réseau en passant par la **liste chaînée** prend le plus de temps et devient très rapidement croissante. Il s'agit de la structure la moins convenable pour reconstituer notre réseau.



1.1 Graphe montrant le temps de calcul de la liste châinée avec les secondes en ordonnée et le nombre de chaînes en abscisses.

En revanche, pour la **table de hachage**, tant que la taille de la table soit supérieure au nombre de chaînes dans le réseau, son temps de calcul n'augmente pas beaucoup. Cela dit, pour notre test, on a décidé que la taille maximale de la table de hachage serait 1000 pour vérifier si dans le pire des cas, ce temps de calcul était supérieur à celui de la liste chaînée.



1.2 Graphe montrant le temps de calcul de la table de hachage et l'arbre quaternaire avec les secondes en ordonnée et le nombre de chaînes en abscisses.

Finalement, **l'arbre quaternaire** se proclame vainqueur car même dans le cas de NbChaines = 5000, son temps de calcul n'atteint jamais une seconde. Nous avons décidé de faire un graphe supplémentaire pour montrer la courte croissance du temps de calcul pour l'arbre quaternaire.

1.3 Graphe montrant le temps de calcul de l'arbre quaternaire avec les secondes en ordonnée et le nombre de chaînes en abscisses.

Pour conclure, on peut dire que si l'espace mémoire n'est pas un facteur important, la table de hachage d'une taille supérieure au nombre de chaînes est l'option la plus convenable. Cependant, en termes de temps de calcul, **l'arbre quaternaire** est la meilleure option pour reconstituer notre réseau de fibres optiques, car sa complexité est en O(nlog4(n)) alors que pour les deux autres structures, la complexité est en $O(n^2)$.